НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем

Кафедра Телекомунікацій

До захисту допущено:

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2021 р.

Дипломна робота

на здобуття ступеня <u>бакалавра</u>

<u>Спеціальності 172 «Телекомунікації та радіотехніка»</u>

на тему: «Управління площиною даних комутаторів мережі SDN за допомогою мов програмування P4 та P4Runtime**»**

Виконав: студент ____4____ курсу, групи __ТЗ-72
(шифр групи)

Топорков Михайло Анатолійович _____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ___професор, к.т.н. Ільченко Михайло Юхимович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультан професор, к.т.н. Романов Олександр Іванович _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент __доцент, к.т.н. Созонник Галина Дмитрівна_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2021 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут телекомунікаційних систем
Кафедра Телекомунікацій

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 172 «Телекомунікації та радіотехніка»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій КРАВЧУК
«___»_____2021 р.

ЗАВДАННЯ
на дипломну роботу студенту
Топоркову Михайлу Анатолійовичу

1. Тема роботи : Управління площиною даних комутаторів мережі SDN за допомогою мов програмування P4 та P4Runtime

керівник роботи ___професор, к.т.н. Ільченко Михайло Юхимович_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 14 квітня 2021 р. № 1007-с

2. Термін подання студентом роботи 7 червня 2021 р.

3. Вихідні дані до роботи: мережі SDN, протокол OpenFlow, мова P4 мова P4Runtime.

4. Зміст роботи:

4.1. Аналіз розвитку програмно конфігурованих мереж

4.2. Архітектура програмно-коніфгурованих мереж

4.3. Потенціал розвитку OpenFlow

4.4. Мова програмування P4

4.5. Архітектура цільового пристрою P4

4.6. Елементи мови P4

4.7. Мова програмування P4Runtime

4.8. Можливості P4Runtime

4.9. Побудова мереж SDN з використанням мови P4

4.10. Пересилання з використанням власного протоколу тунелю

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Презентація по темі “ Управління площиною даних комутаторів мережі SDN за допомогою мов програмування P4 та P4Runtime ” в кількості ** слайдів

5.1 Архітектура мережі SDN

5.2 Протокол Openflow

5.3 Підходи до програмованості мікросхем

5.4 Мова програмування P4

5.6 Архітектура цільового пристрою P4, його принцип програмування та основні елементи мови P4

5.7 Мова програмування P4Runtime

5.8 Мережа з використанням мови P4

6. Дата видачі завдання 06.02.2021

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Вступ	09.02.2021-02.03.2021	виконано
2.	Аналіз архітектури SDN, та протоколу OpenFlow як API для взаємодії площини управлянні з площиною даних	08.03.2021-23.03.2021	виконано
3.	Пошук інформації про мови програмування P4 та P4Runtime	25.03.2021-01.04.2021	виконано
4.	Аналіз процесу обробки пакетів в цільових пристроях P4	05.04.2021-18.04.2021	виконано
5.	Аналіз можливостей мови з використанням платформи Mininet	22.04.2021-16.05.2021	виконано
6.	Висновки	31.05.2021-04.06.2021	виконано

Студент Михайло, ТОПОРКОВ

Керівник Михайло, ІЛЬЧЕНКО

РЕФЕРАТ

Дипломна робота містить: 57 сторінок, 18 рисунків, 2 таблиці, 6 лістингів, 18 посилань.

З стрімким розвитком програмно конфігурованих мереж (SDN) вимоги до їх гнучкості також зростають. Це призводить до пошуку нових видів програмованості в яких залежність від фізичного функціоналу мікросхем є значно меншою.

Метою даної роботи є аналіз мови програмування для мережевих пристроїв P4. Демонстрація основних можливостей мови та дослідження можливості заміни протоколу OpenFlow на P4 та порівняння з нею.

В дипломній роботі проведено аналіз архітектури SDN, взаємодію площини управління з площиною пересилання даних з використанням протоколу OpenFlow. Розглянуто мову програмування P4 та P4Runtime, досліджено основні характеристики мов та їх основні елементи. Продемонстровано застосування P4 для визначення свого власного протоколу тунелю.

Ключові слова: SDN, OpenFlow, P4, P4Runtime.

ABSTRACT

The diploma work consists of 57 pages, 18 figures, 2 tables, 6 listings, 18 links.

With the rapid development of software-configured networks (SDNs), the requirements for their flexibility are also growing. We can offer a search for new types of programmability depending on the physical functional microchemistry.

The purpose of this work is to analyze the programming language for network devices P4. Demonstration of the main features of the language and research of the possibilities to replace the OpenFlow protocol with P4 and comparison with it.

The thesis analyzes the SDN architecture, the interaction of plane control with the plane of data transmission using the OpenFlow protocol. The programming language R4 and P4Runtime is considered, the main characteristics of languages and their main elements are investigated. The use of P4 to determine the tunnel's own protocol is demonstrated.

Keywords: SDN, OpenFlow, P4, P4Runtime.

Зміст

Список скорочень

API	Application Programming Interface
ONF	Open Networking Foundation
P4	Programming Protocol-independent Packet Processors
PISA	Protocol Independent Switch Architecture
QoS	Quality of Service
RAN	Radio access network
RTC	Real Time Communications
SDN	Software-Defined Networking
TLS	Transport Layer Security
VoIP	Voiceover Internet Protocol
SAI	Switch Abstraction Interface
BMv2	Behavioral Model V2

Вступ

Стрімкий розвиток програмно-конфігурованих мереж (SDN) започаткував тенденцію відділення площини управління мережею від площини даних, управління мережею стало не пов’язано з мережевим апаратним забезпеченням, на якому вона працювала.

Метою програмно-конфігурованих мереж є надання можливості інженерам та адміністраторам хмарних обчислень та адміністраторам швидко реагувати на зміни вимог ринку за допомогою централізованої консолі управління. SDN охоплює безліч видів мережевих технологій, розроблених для того, щоб зробити мережу більш гнучкою та гнучкою для підтримки віртуалізованих серверів та інфраструктури зберігання сучасних центрів обробки даних.

Але при все вище сказане основні інновації в технологіях припадали на площину управління, на площину даних де фактично пересилаються мережеві пакети припадало менше інновацій. В останні роки була розроблена нова мова програмування для комутаторів під назвою «P4».

P4 - мова високого рівня для програмування площини даних мережевих комутаторів. Комутатори які підтримуються P4 не мають заздалегідь визначеної поведінки або протоколів. Тоді як в даний час більшість мережевих комутаторів будуються за підходом знизу вгору «Bottom-up», це комутатори з фіксованим набором функцій, що мають один спосіб переадресації пакетів. В свою чергу комутатори з підтримкою P4 побудовані за підходом зверху вниз «Top-Down». Це надає можливість користувачу визначати функціонал який буде наявний, а також виключати небажаний. В результаті гнучкість програмно-конфігурованих мереж зростає що дозволяє адаптувати мережу під передачу різних видів трафіку.

1. Аналіз розвитку програмно конфігурованих мереж

1.1. Передумови появи концепції SDN

На даний момент ряд експертів характеризують поточну ситуацію в мережевий галузі як «критичну і революційну». Домінація на ринку представлена закритими, пропрієтарними рішеннями для додатків, а сумісність рішень різних вендорів забезпечується в кращому випадку на рівні інтерфейсів. Мережі є надто складними, що ускладнює їх масштабування і управління ними, та знижує їх надійність. Очевидно, що це гальмує подальший розвиток мереж і додатків що функціонують в них.

Основними передумовами до появи концепцій програмно-конфігурованих мереж (Software-Defined Networking, SDN) є, перш за все, швидкий ріст трафіку даних і кількості підключених до мережі пристроїв.

При цьому сам трафік стає різнорідним - якщо в кінці 1990-х рр. його основу становила пересилання даних і файлів, які не потребують особливих вимог до каналу, за винятком швидкості передачі даних, то вже до середини 2000-х на перше місце вийшли питання забезпечення якості сервісу (Quality of Service, QoS), мінімальної затримки в каналі (latency) і ін. Це, в першу чергу, пов'язано зі зміною структури призначеного для користувача трафіку, в якому стали переважати комунікації в реальному часі (Real Time Communications, RTC), IP телефонія (Voiceover Internet Protocol, VoIP), відеосервіси тощо. У операторів виникла реальна потреба в динамічної пріоритетності трафіку. Наприклад, в деяких випадках пріоритет повинен бути зроблений для FTP протоколу, в інших - для SIP і навпаки.

В області мобільного зв'язку установка додаткових макростільників (базових станцій) після досягнення певного порогу щільності їх розміщення вже не дає істотного приросту пропускної спроможності і ємності мереж радіодоступу (Radio access network, RAN), тому наступним етапом стає використання малих

стільників (фемто- і пікостільників). В результаті конфігурація великомасштабних мереж перетворюється на складне завдання і вимагає серйозних змін принципів побудови, експлуатації та управління мереж та управління ними.

1.2. Архітектура програмно-коніфгурованих мереж

Програмно-конфігурована мережа (Software-Defined Networking, SDN) виникла як мережева парадигма, яка відокремлює площину пересилання даних (Data plane) від площини управління (Control Plane) за рахунок централізації стану мережі та можливостей прийняття рішень у площині управління. Яка представлена контролером SDN, залишаючи просту операцію пересилання на площині даних, мережевих пристроях SDN та абстрагування базової мережевої інфраструктури до площини додатків. Розділення площини управління та площини пересилання даних здійснюється через програмований інтерфейс (Application Programming Interface, API) між мережевими пристроями SDN та контролером SDN.

Open Networking Foundation (ONF) визначає архітектуру високого рівня для SDN [1] з трьома основними рівнями, як показано на рисунку 1.1. Рівень додатків складається з додатків для кінцевих користувачів, які використовують комунікаційні та мережеві послуги SDN. За допомогою контролера додатки можуть впливати на поведінку базової інфраструктури, налаштовуючи потоки таким чином, щоб маршрутизувати пакети через найкращий шлях між двома кінцевими точками, балансуючи навантаження трафіку через кілька шляхів або призначених до набору кінцевих точок, реагуючи на зміни в топології мережі, такі як збій каналів та додавання нових пристроїв та шляхів, або перенаправлення трафіку з метою перевірки, автентифікації, сегрегації та подібних завдань, пов’язаних із безпекою. Рівень управління забезпечує логічно централізовану функціональність управління, яка контролює поведінку мережевої переадресації

через відкритий інтерфейс. Контролер SDN контролює через API всі пристрої SDN, що складають мережеву інфраструктуру.

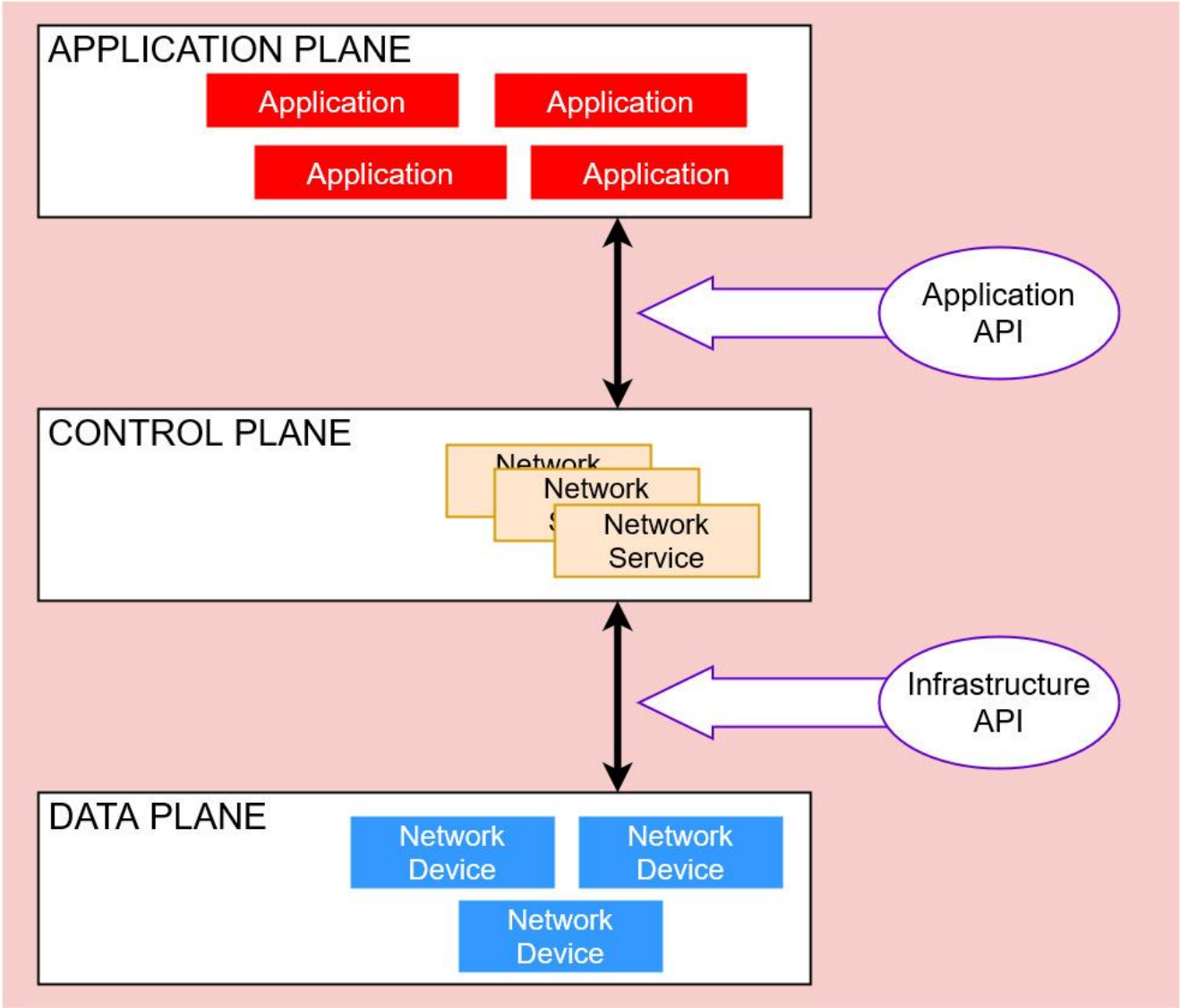

Рис. 1.1. Архітектура SDN мережі

Та реалізовує політичні рішення, такі як: маршрутизація, переадресація, балансування навантаження тощо. Це забезпечує абстрактний вигляд усієї мережі для додатків через інтерфейс інфраструктури. Рівень передачі даних або рівень інфраструктури складається з пристроїв SDN (як фізичних, так і віртуальних), які виконують комутацію та переадресацію пакетів. Зокрема, пристрій SDN складається з API для зв'язку з контролером, рівня абстракції та компонента обробки пакетів. Рівень абстракції абстрагує пристрій SDN як набір таблиць потоків. Функція обробки пакетів приймає рішення про дії, які слід виконати, на

основі результатів оцінки вхідних пакетів щодо записів потоку в таблицях потоків.

1.3. Протокол OpenFlow

Для взаємодії площини передачі даних з площиною управління, ONF було створено стандартизований протокол який отримав назву OpenFlow [2]. OpenFlow визначає зв'язок між контролером та комутатором OpenFlow. Повідомлення зв'язку між ними передаються через захищений канал, який реалізований через з'єднання TLS (Transport Layer Security) через TCP. За допомогою обміну командами та пакетами, контролер визначає та програмує поведінку переадресації пакетів комутатора. Відповідно комутатор виконує переадресацію пакетів і повідомляє про стан та вид трафіку.

Користувальницький трафік класифікується на потоки на основі його характеристик. Комутатор OpenFlow виконує пошук та перегляд пакетів відповідно до потоку, якому вони належать. Потік - це набір пакетів, переданих від однієї кінцевої точки мережі (або набору кінцевих точок) до іншої кінцевої точки мережі (або набору кінцевих точок). Кінцеві точки можуть бути визначені як пари адрес IP-TCP / UDP, кінцеві точки VLAN або порти вводу-виводу комутатора.

Для контролера, комутатор OpenFlow представлений набором таблиць потоків. Таблиця потоків складається із записів потоку. Запис потоку містить поля заголовків, поля лічильників та поля дій. Поля заголовків використовуються для ідентифікації потоку пакета, поля лічильника використовуються для збору статистики ідентифікованого потоку, а поля дій містять інструкції та дії, які комутатор повинен виконувати над пакетами цього потоку рисунок 1.2.

Контролер на основі своїх знань про базову топологію мережі, можливості пристрою та вимоги до програми визначає потік з відповідними полями та використовує протокол OpenFlow для програмування комутатора з необхідними таблицями та записами потоків.

OpenFlow визначає три типи повідомлень: від контролера до комутатора (controller-to-switch), асинхронні (asynchronous), та симетричні (symmetric) повідомлення.

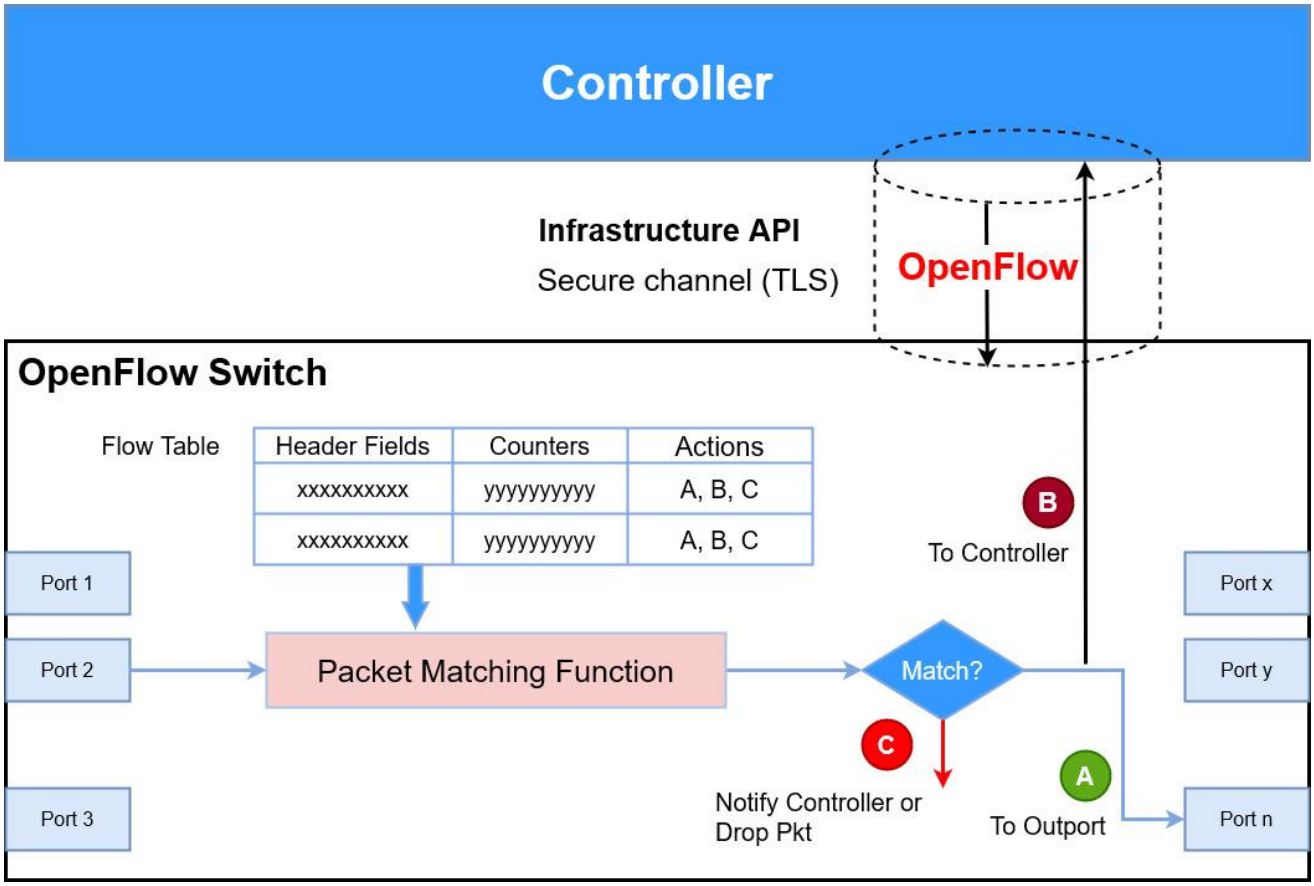

Рис. 1.2. Основі операції комутатора OpenFlow

Повідомлення від контролера до комутатора використовуються для управління та програмування комутатора (наприклад, налаштування конфігурації комутатора, надсилання таблиць потоків, запит статистики трафіку). Асинхронні повідомлення надходять від комутатора до контролера, без попереднього запиту контролера. Вони використовуються для сповіщення контролера про зміни в стані комутатора та для повідомлення про мережеві події, включаючи помилки. Симетричні повідомлення використовуються як комутатором, так і контролером для визначення активності з'єднання.

Основні функції комутатора OpenFlow включають взаємодію з контролером за допомогою протоколу OpenFlow, ідентифікацію потоків трафіку за допомогою

зіставлення пакетів, виконання переадресації пакетів та звітування статистики та стану перемикання на контролер.

1.4. Конвеєр обробки пакетів

OpenFlow-конвеєр складається з багатьох потокових таблиць у кожній таблиці багато потокових записів. Специфікація протоколу OpenFlow вимагає, щоб комутатор мав би хоч одну потокову таблицю, хоча можливо мати і кілька таблиць.

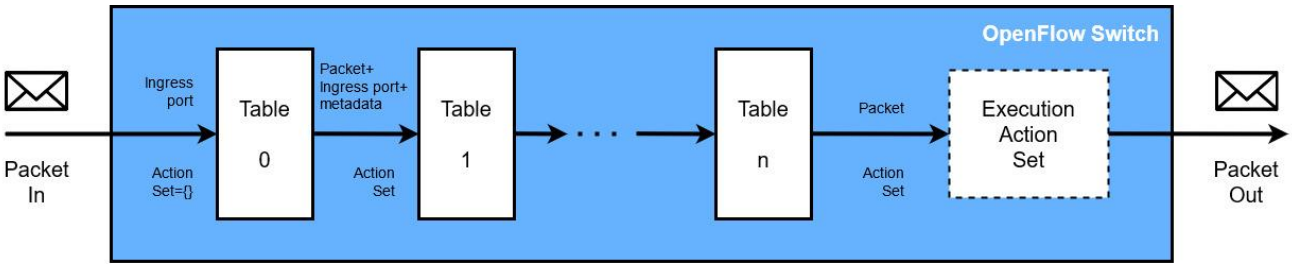

Рис. 1.3. Діаграма конвеєру обробки пакетів OpenFlow

На рисунку 1.3 зображено конвеєр обробки мережевих пакетів OpenFlow. Потокові таблиці пронумеровані послідовно починаючи від 0. Обробка завжди починається з початкової таблиці: заголовок пакета порівнюється з полями потокових записів таблиці 0. Якщо знайдений відповідний потоковий запис, виконуються інструкції, визначені в даному записі. Для подальшої обробки, пакет можна передавати тільки в таблицю з великим порядковим номером, ніж поточний. В останній таблиці, в якості інструкцій заборонено використовувати передачу далі по конвеєру (дозволений тільки список безпосередніх дій над пакетом). Якщо в знайденому потоковому запису немає інструкцій передачі пакета далі по конвеєру, обробка на цьому місці закінчується і до пакету застосовуються дії, додані в action set в процесі обробки.

Якщо жодний запис не підходить для оброблюваного пакета, генерується подія table miss. Дія застосована до пакету залежить від статичної конфігурації

комутатора (можна скинути пакет, відправити на контролер або відправити на обробку в спеціальну таблицю).

1.5. Потенціал розвитку OpenFlow

Протокол OpenFlow починався з простого, з абстракції єдиної потокової таблиці, яка могла порівнювати пакети за їхніми заголовками. За останні роки специфікація стала все більш складною таблиця 1, з набагато більшою кількістю заголовків і декількома етапами потокових таблиць, що дозволяє комутаторам надавати контролерам більше своїх можливостей.

Табл. 1 Покоління специфікацій OpenFlow

Version	Date	Header Fields
OF 1.0	Dec 2009	12 Fields (Ethernet, TCP/IPv4)
OF 1.1	Feb 2011	15 Fields (MPLS,inter table metadata)
OF 1.2	Dec 2011	36 Fields (ARP, ICMP, IPv6, ect)
OF 1.3	Jun 2012	40 Fields
OF 1.4	Oct 2013	41 Fields
OF 1.5	Mar 2015	41 Fields

Збільшення кількості нових заголовків не припиняється. Наприклад, оператори мереж центрів обробки даних все частіше хочуть застосовувати нові форми інкапсуляції пакетів (наприклад, NVGRE, VXLAN і STT), для чого вони вдаються до розгортання програмних комутаторів, які легше розширити за допомогою нових функцій. Але OpenFlow не здатний надати бажану гнучкість, таку як зміну процесу обробки пакетів в комутаторах або додавання нових протоколів безпосередньо власниками мережевого обладнання.

Останні розробки мікросхем демонструють, що така гнучкість може бути досягнута в спеціалізованих ASIC на терабітних швидкостях [3, 4]. Програмувати

це нове покоління мікросхем комутаторів непросто. Кожен чіп має свій власний низькорівневий інтерфейс, схожий на програмування мікрокода. Тому шукаються нові підходи до програмування мікросхем, які в свою чергу приносять нові спеціалізовані мови програмування.

1.6. Новий підхід до програмованості

Зазвичай мережеві пристрої, такі як комутатори або маршрутизатори, найчастіше розробляються за підходом «Bottom-up», тобто знизу вгору. Спочатку проектується мікросхема, котра є серцем системи і на практиці визначає функціональність мережевого пристрою. Оскільки мікросхема є фіксованою одиницею то конвеєр обробки пакетів та протоколи які підтримуються не можуть бути легко зміненими. Додавання нових функцій є складним і затратним процесом, який може зайняти місяці. Це пояснюється тим, що, як правило, потрібно перепроектовувати мікросхему.

Для досягнення програмованості мікросхем в мережевих пристроях було запропоновано використання нового підходу, замість підходу bottom-up. Реалізацією такого підходу стала мова P4 [5].

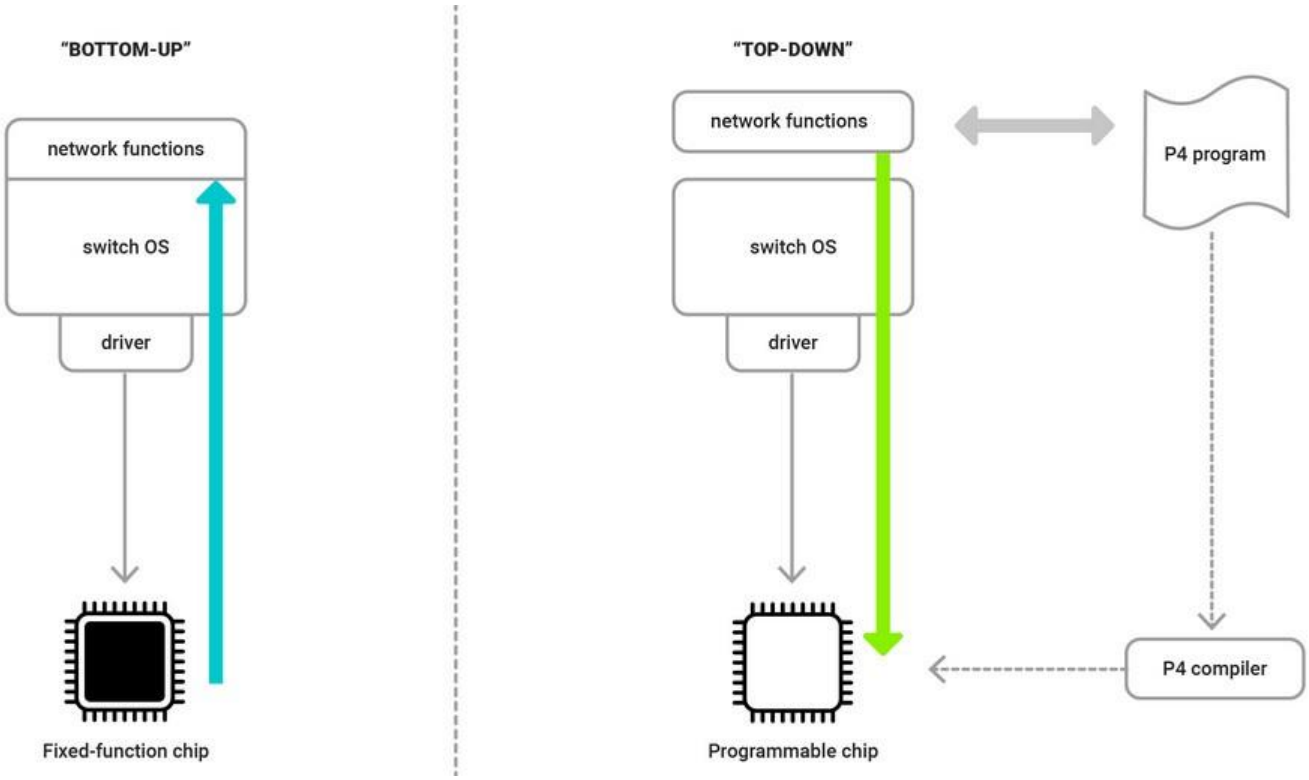

Рис. 1.4. Підходи до програмованості «Bottom-up» та «Top-down»

P4 представляє підхід, подібний тому, як працює Центральний Процесор (ЦП) або Графічний Процесор (ГП). Ці одиниці обробки виконують код, написаний певною мовою програмування (наприклад, C ++ для процесора або OpenCL для графічного процесора). Код спочатку компілюється, а потім завантажується в процесор. P4 заснований на тому ж принципі, але для мережевих пристроїв. Цей підхід називається «Top-down». На відміну від підходу знизу вгору, тут програміст визначає набір функцій мережі. Цей набір описано в програмі P4. Потім код компілюється і конфігурація завантажується в мережевий пристрій рисунок 1.4.

Підхід зверху вниз робить площину пересилання даних повністю програмованою. Тепер мережевий пристрій можна безпосередньо запрограмувати, таким чином щоб отримати бажану поведінку пересилання та обробки пакетів. Це позбавляє від необхідності статичного протоколу OpenFlow.

Щоб використовувати даний підхід потрібні спеціалізовані мікросхеми, в свою чергу такі мікросхеми все ще можуть мати фіксовані частини, але переважно

більша кількість повинна є програмованою. Повна програмованвсть частіше всього досягається з використання програмного мережевого обладнання.

Висновки до 1-го розділу.

В цьому розділі було розглянуто концепцію та архітектуру мереж SDN та їх особливості.

Також розглянуто протокол OpenFlow як інтерфейс для взаємодії площин управління та пересилання даних. Розглянуто його принципи роботи та недоліки в сучасних вимогах до програмованості. Розглянуто новий підхід до програмованості мережевих пристроїв з використанням P4, який потенційно може стати заміною протоколу OpenFlow в найближчому майбутньому.

2. Структура і основні елементи мови програмування P4

2.1. Мова програмування P4

P4, скорочення від Programming Protocol-independent Packet Processors - це мова програмування, призначена для опису процесу обробки пакетів в площині даних комутаторів. Вперше мова була представлена в документі SIGCOMM CCR під назвою «P4: Programming Protocol-Independent Packet Processors» [6]. Спочатку мова P4 була створена групою інженерів і дослідників з Google, Intel, Microsoft Research, Barefoot, Princeton і Stanford. Мета була проста: створити просту у використанні мову, яку розробник програмного забезпечення зможе вивчити за день, і використовувати для точного опису того, як пакети обробляються в мережі. Оскільки мова призначена для пристроїв які відповідають за

маршрутизацію, список вимог і варіантів дизайну відмінний порівняно з мовами програмування загального призначення. Розробники мови виділили три основні риси:

1. Незалежність від цільової реалізації;
2. Незалежність від використовуваного протоколу (-ів);
3. Реконфігурованість полів.

Незалежність від цільової реалізації. Програми P4 розробляються так, щоб вони не залежали від реалізації, тобто вони можуть бути скомпільовані для безлічі різних типів виконавчих машин, таких як процесори загального призначення, FPGA, системи на кристалі, мережеві процесори і ASIC. Ці різні типи машин відомі як цільовий пристрій P4, і під кожний цільовий пристрій необхідний компілятор для перетворення вихідного коду P4 в модель цільового комутатора. Компілятор може бути вбудований в цільовий пристрій, зовнішнім програмним забезпечення або навіть хмарним сервісом. Оскільки багато початкових цільових пристроїв для програм P4 використовувалися для простої комутації пакетів, дуже часто можна почути термін «комутатор P4», навіть якщо вживання «цільовий пристрій P4» найчіткіше.

Незалежність від використовуваного протоколу (-ів). P4 незалежний від протоколів. Це означає, що мова не має вбудованої підтримки поширених протоколів, таких як IP, Ethernet, TCP, VxLAN або MPLS. Замість цього програміст P4 описує формати заголовків і імена полів необхідних протоколів в програмі, які в свою чергу інтерпретуються і обробляються компільованою програмою і цільовим пристроєм.

Реконфігурованість полів. Незалежність від протоколу і модель абстрактної мови допускають реконфігурованість – цільові пристроїв P4 повинні мати можливість змінювати обробку пакетів після розгортання системи. Ця можливість традиційно пов'язана з маршрутизацією за допомогою процесорів загального

призначення або мережевих процесорів, а не інтегральних схем з фіксованими функціями.

Хоча в мові немає, того щоб могло перешкодити оптимізувати роботу певного набору протоколів, ці оптимізації невидимі для автора мови і можуть, в кінцевому підсумку, знизити гнучкість системи і цільового пристрою і їх реконфігурованість. Дані характеристики мови спочатку закладалися її творцями з орієнтацією на повсюдне використання її в мережевій інфраструктурі.

2.2. Місце P4 в архітектурі SDN

Багато комутаторів мають, як реалізацію площини пересилання даних так і площини управління, комутатори OpenFlow в свою чергу реалізують тільки площину пересилання даних. P4 призначений лише для опису процесу обробки пакетів в площині даних комутаторів. Програми P4 також частково визначають інтерфейс, за допомогою якого здійснюється взаємодія площини управління та площини даних, але P4 не може використовуватися для опису функціональності площини управління комутатора. На рисунку 2.1 зображено різницю між звичайним комутатором та P4 програмованим комутатором з точки зору архітектури SDN.

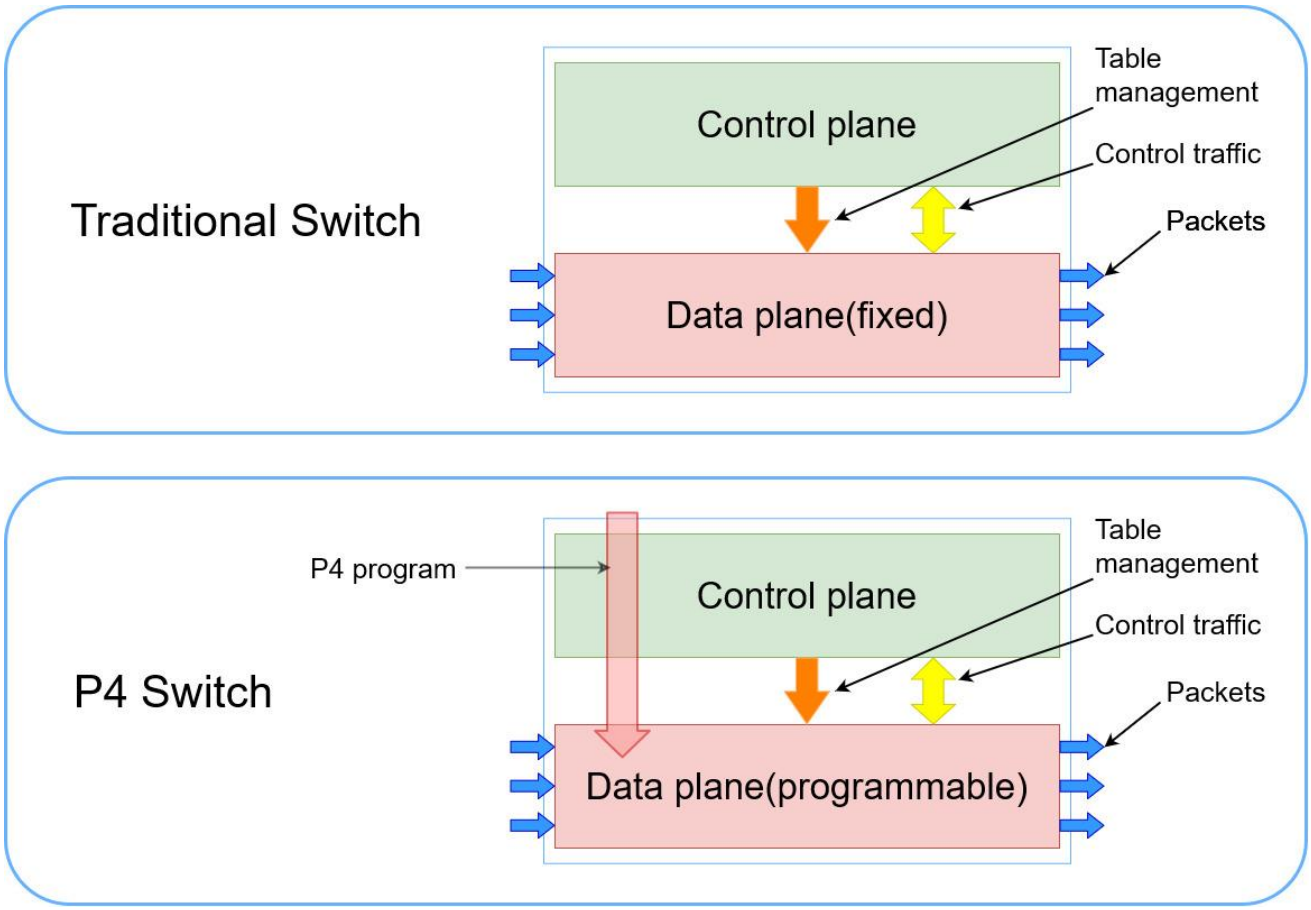

Рис. 2.1. Звичайний та P4 програмований комутатори

В звичайному комутаторі, функціональність площини пересилання даних визначається виробником комутатору. Площина управління керує площиною даних, керуючи записами в таблицях, конфігуруючи спеціалізовані об'єкти, наприклад лічильники та обробляючи пакети управління. Комутатори OpenFlow також мають чітко визначену функціональність площини пересилання даних, котрою керує площина управління. P4 програмований комутатор відрізняється від звичайного та OpenFlow комутаторів двома основними шляхами:

По-перше, функціональність площини пересилання даних не визначається виробником комутатору, тобто вона не є фіксованою а визначається програмою P4. Площина пересилання даних налаштовується під час ініціалізації для реалізації функціональних можливостей, описаних в програмі P4, і не має вбудованих знань про існуючі мережеві протоколи.

По-друге, площина управління взаємодіє з площиною даних за допомогою тих самих каналів, що і у звичайному комутаторі, але набір таблиць та інших об’єктів у площині даних більше не фіксовані, оскільки вони визначаються програмою P4. Компілятор P4 генерує API, який використовується площиною управління для зв'язку з площиною даних.

2.3. Архітектура цільового пристрою P4

Подібно до того, як програми на мові C компілюється за певною архітектурою для перетворення загальних C-конструкцій в конструкції, специфічні для архітектури, програми P4 пишуться під конкретні архітектури. Protocol Independent Switch Architecture (PISA) є прикладом архітектури, на якій може працювати P4 рисунок 2.2. Її апаратна структура має чітке розділення між синтаксичним аналізом, виконанням процесу обробки за таблицями та відновлення пакету.

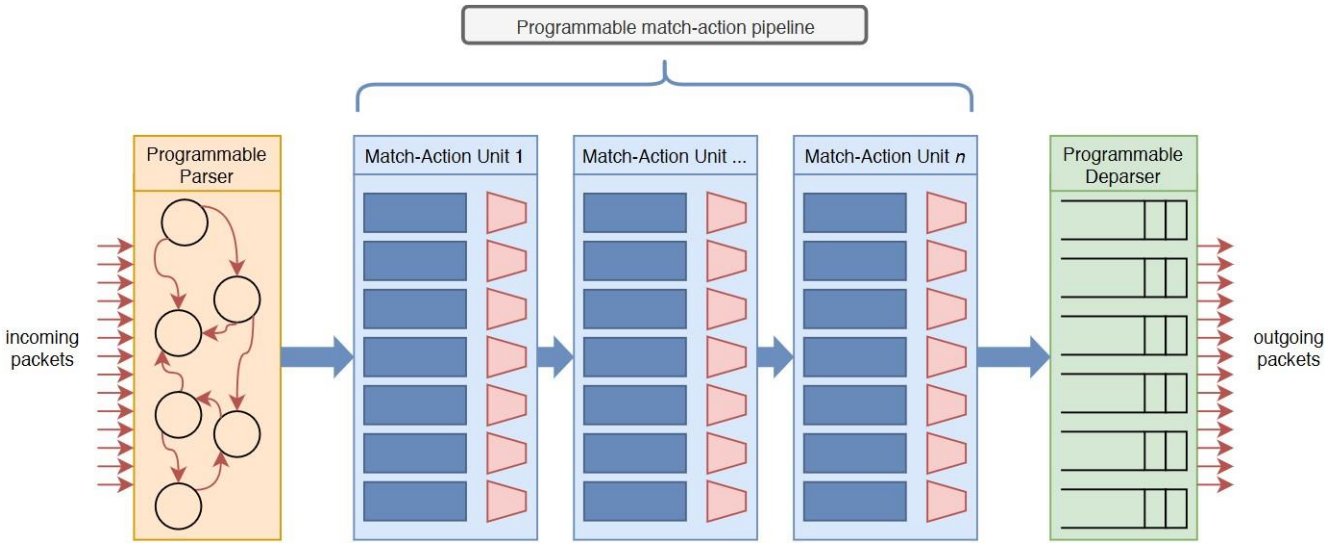

Рис. 2.2 Архітектура PISA

PISA - не єдина архітектура, з якою можуть працювати програми P4. Виробник пристрою з підтримкою P4 постачає компілятор, який компілює загальний код P4, щоб створити двійковий файл, який запустить конкретний

цільовий пристрій. Разом із компілятором, виробник постачає опис архітектури. Архітектура точно описує структуру апаратного забезпечення та наявні компоненти. Таким чином, апаратні модулі, які не обов'язково присутні на всіх пристроях, все ще можуть використовуватися. Наприклад, опис архітектури може описувати наявність модуля контрольної суми апаратного забезпечення. Ці модулі визначені зовнішніми бібліотеками і доступні з коду через зовнішні об'єкти. Офіційна специфікація P4 визначає базовий набір інструкцій, які повинні виконувати всі пристрої, додаткова функціональність забезпечується через зовнішні об'єкти, зазначені в описі архітектури та зовнішніх бібліотеках. Цей опис також визначає частини та параметри, які офіційна специфікація залишає відкритими для пристроїв. Наприклад, максимально підтримувана бітова ширина може бути різною на різних пристроях, кожен пристрій може встановити свій власний максимум. Такі значення описані в описі архітектури рисунок 2.3.

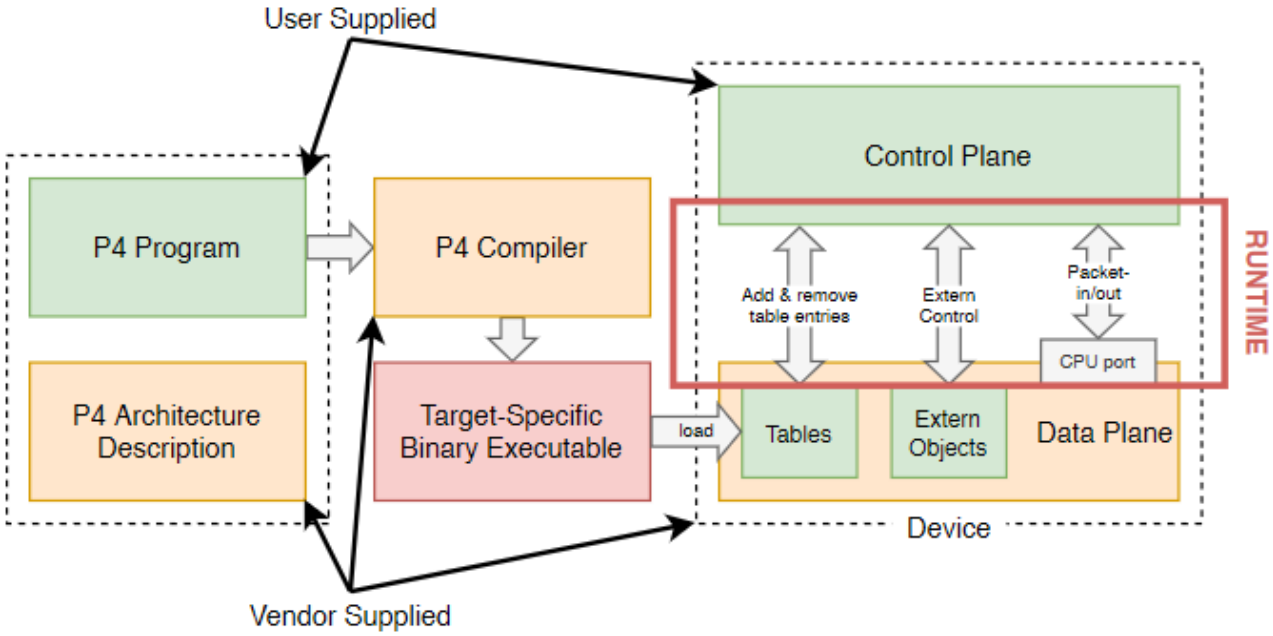

Рис. 2.3 Опис архітектури

2.4. Елементи мови P4

Мова P4 є мовою програмування високого рівня з синтаксисом, подібним до C. Оскільки P4 є дуже специфічною в ній немає багато конструкцій які наявні в

мовах загального призначення, таких як C або Java, наприклад, цикли або довільні масиви. На рисунку 2.4. зображено опис основних елементів які містить програма P4 та де вони знаходяться на прикладі архітектури PISA.

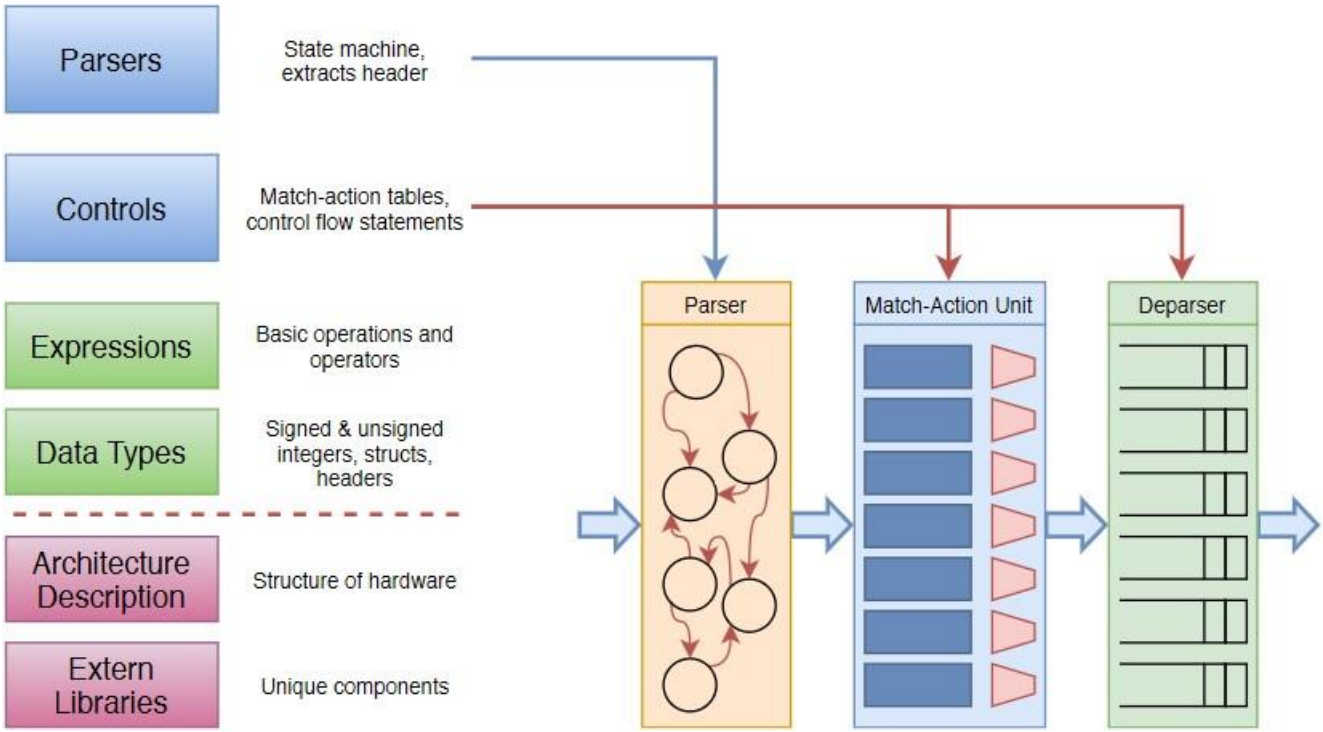

Рис. 2.4 Місця компонентів в архітектурі PISA

Мова P4 побудована на двох основних компонентах: логікі парсеру та логікі конвеєра обробки. Логіка парсеру побудована як машина яка розбиває пакет за відповідними заголовками та витягає небідні поля із заголовків. Логіка конвеєра обробки це таблиці зіставлення, які в певній послідовності утворюють конвеєр обробки пакетів.

2.4.1. Заголовки та парсер

Заголовки P4. P4 не підтримує жодного конкретного протоколу а також не містить інформації про протоколи які використовуються в мережі. Замість цього, програміст повинен точно вказати поля які наявні в заголовку. За допомогою полів визначається структура заголовка, тобто його розмір та порядок полів. В лістингу 1 наведено приклад заголовків протоколів Ethernet та IPv4. Заголовок

Ethernet містить три поля: два 48-розрядні широкі адреси та один 16-бітний рядок який вказує тип протоколу наступного рівня. В свою чергу заголовок IPv4 містить

```
typedef bit<9>   egressSpec_t;

typedef bit<48>  macAddr_t;

typedef bit<32>  ip4Addr_t;

header ethernet_t{

    macAddr_t  dstAddr;
    macAddr_t  srcAddr;
    bit<16>    etherType;


}

header IPv4_t {
    bit<4>       version;
    bit<4>       ihl;
    bit<8>       tos;
    bit<16>      totalLen;
    bit<16>      identification;
    bit<3>       flags;
    bit<13>      fragOffset;
    bit<8>       ttl;
    bit<8>       protocol;
    bit<16>      hdrChecksum;
    ip4Addr_t    srcAddr;
    ip4Addr_t    dstAddr;
}
struct headers{
    ethernet_t ethernet;
    IPv4_t     ipv4;
}
```

всі поля які відповідають даному протоколу [8].

Лістинг 1. Приклад заголовку протоколу Ethernet та IPv4

Також P4 підтримує використання власних полів в заголовках, вони визначаються за допомогою ключових слів «typedef» або «type», щоб зробити код легшим для читання та підтримки. В кінці прикладу наявна структура яка містить всі заголовки, які будуть відправлені на аналіз.

Парсер. Пакети надходять до цільового пристрою P4 як потік байтів. Враховуючи попереднє визначення заголовку пакету, програма P4 тепер знає, як

інтерпретувати потік байтів. Парсер як абстрактна машина використовується для поступового розбивання цього потоку і витягування заголовків, які містять значення вхідного пакета. Парсер завжди починається з стану «start». Заголовки витягуються за допомогою структури «extract( )», переходи визначаються за

```
parser MyParser(packet_in packet,
           out headers hdr,
           inout metadata meta,
           inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select (hdr.ethernet.etherType) {
           TYPE_IPV4: parse_ipv4;
           default: accept;
        }
    }
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

допомогою ключового слова transition.

Лістинг 2. Приклад парсеру

Поля, витягнуті з заголовків, можуть бути використані для умовного керування переходами, як правило, через структуру «select()». Це дозволяє парсерам також розбивати та аналізувати власні заголовки. В лістингу 2 зображено, як заголовки Ethernet та IPv4, які наведено в лістингу 1, можуть бути витягнуті для подальшого використання. Значення заголовка можуть бути використані для керування логікою програми. Наприклад на цільовий пристрій надходить пакет, значення поля etherType витягується з заголовку Ethernet, та приймається (на даному етапі

можливо вказати значення при якому etherType буде прийматися, наприклад 0x0800 для протоколу IPv4. За замовчуванням строїть значення прийняти). Після чого починається аналіз заголовку IPv4 цього пакету. Заголовок витягується і приймається. В парсері пакет може бути прийнятий, оброблений, і відправленій або відхилений і скинутий.

2.4.2. Дії та конвеєр обробки пакетів

Дії. Обчислювальна логіка дій P4 міститься в структурах які визначаються ключовим словом «action». Ці дії містять імперативний код, який виконує фактичну логіку та маніпуляції з полями заголовків пакетів. Дії можуть бути використані, серед іншого, для зменшення поля Time-To-Live (TTL) у заголовку IP або для встановлення наступної адреси переходу на основі адреси призначення пакета. В лістингу 3 наведено приклад дії, яка зменшує поле TTL і встановлює

```
action drop() {
    mark_to_drop(standard_metadata);
}

action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port;

    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

    hdr.ethernet.dstAddr = dstAddr;

    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;

}
```

адресу наступного пункту призначення пакету.

Лістинг 3. Приклад дій.

Конвеєр обробки пакетів. Конвеєр обробки пакетів представляє з себе набір таблиць через які проходить пакет. Ці таблиці є загальним узагальненням традиційних таблиць комутаторів. Вони забезпечують зв'язок між певним

```
table ipv4_lmp {
   key = { hdr.ipv4.dstAddr: lmp; }
   actions = { ipv4_forward; drop; NoAction; }
   size = 1024;

   default_action = drop();
}
```

ключем, тобто та відповідною дією.

Лістинг 4. Приклад таблиці.

Цей ключ може бути взятий з різних джерел, таких як: поля в заголовках, метадані, які програма P4 відстежує разом із заголовком, або навіть зовнішні об'єкти. Наприклад в лістингу 4 наведено приклад таблиці для маршрутизації за IPv4.

В таблиці визначено дві можливі дії, які можуть бути застосовані до пакету: перша – скинути; друга – встановити наступну адресу пересилання, включаючи встановлення відповідного інтерфейсу вихідного порту. Ключ - це останній префікс, який відповідає адресі призначення. Сама таблиця заповнюється контролером або за допомогою командної строки пристрою. Контролер заповнює таблицю записами, які пов'язують певний ключ із конкретною дією. У цьому прикладі ключі будуть прив'язані до дії «ipv4_forward ()», тоді як невизначені ключі неявно прив'язані до дії «drop()». P4 позначає всю таблицю як єдину одиницю збігу (match-action unit).

Багато з цих одиниць дії збігу можуть бути зв'язані між собою, щоб сформувати складну поведінку та логіку. Такий конвеєр обробки пакетів визначається за допомогою структури «control». Цей елемент керування визначає, які дії та таблиці визначені та в якому порядку вони застосовуються лістинг 5.

```
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standart_metadata_t standart_metadata) {

       action drop() { … }

       action ipv4_forward(…) { … }

       table ipv4_lmp() { … }

       …

       apply {
           if (hdr.ipv4.isValid()) {

                ipv4_lmp.apply();

         }
      }
}
```

Лістинг 5. Структура control

На рисунку 2.4 показано, як такий ланцюжок може бути використаний у простому пристрої переадресації IPv4. У цьому потоці є кілька точок, в яких пакет може бути скинутий, наприклад, якщо сталася помилка парсера. Потім, на основі адреси призначення IPv4, встановлюється наступний перехід та порт виведення, а TTL зменшується. У наступній таблиці перевіряється TTL. Потім MAC-адреса призначення встановлюється в кадрі Ethernet. Нарешті, MAC-адреса джерела встановлюється як поточний комутатор.

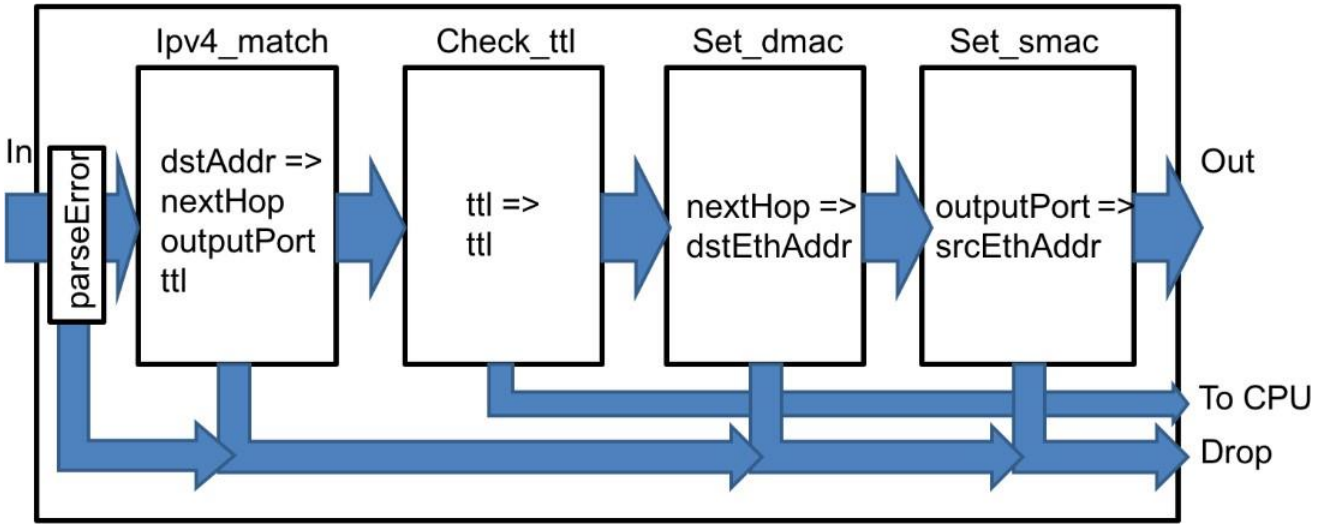

Рис. 2.5 Діаграма конвеєру обробки пакетів виражена мовою P4

2.4.3. Депарсер

Програма P4 також повинна повторно зібрати пакет, перш ніж цей пакет може бути введений а фізичний носій. Для цього P4 також використовує структуру «control». Подібно до того, як заголовки витягуються в парсері методом «packet.extract ()», пакети збираються за допомогою методу «packet.emit ()». Цей процес може також включати певну логіку, наприклад, встановлення контрольної суми на нове значення. В лістингу 6 наведено приклад депарсера, який збирає та

```
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

видає заголовки IP та Ethernet.

Лістинг 6. Приклад депарсеру

2.5. Потенціал використання P4

На даний момент P4 знаходиться на початковій стадії прийняття, але він вже демонструє позитивну динаміку інтересу серед гравців ринку. Організації, що мають достатньо внутрішнього досвіду для програмування мережевих чіпів, тепер можуть самі визначати функціонал площини пересилання даних. Їм більше не потрібно чекати, поки їхній постачальник ASIC додасть новий протокол або змінить певний параметр, це можна зробити за допомогою внутрішніх інновацій.

Без сумніву гнучкість що надає P4 потенційно збільшує економічну ефективність. На даний момент мова уже використовується в багатьох компаніях:

Китайська компанія Tencent є найбільшою інвестиційною компанією в світі. Дочірні компанії Tencent, як в самому Китаї, так і в інших країнах світу, спеціалізуються на різних областях високотехнологічного бізнесу, в тому числі різних інтернет-сервісах, розробках в області штучного інтелекту і електронних розваг. P4 і програмована маршрутизація є передовими технологіями, які використовуються в архітектурі мережі компанії.

Компанія Google будучи одним із засновників з гордістю відзначає швидке впровадження P4 в мережевій індустрії і, зокрема, в сфері архітектурного проектування ЦОД.

Компанія AT&T була однією з перших прихильників P4, одна з перших використала P4 для визначення поведінки, яку хотіла бачити в мережах, і використовує пристрої програмованої переадресації P4 в своїй мережі.

Xilinx був одним із засновників P4.org і брав активну участь в розробці мови P4 і впровадили її в програмовані платформи на основі FPGA для обладнання SmartNIC і NFV, випустивши один з перших компіляторів $P4_{16}$ в якості частини дизайну SDNet [9].

Таким чином, P4 - це незалежна від цільової реалізації та протоколів мова програмування, яка вже використовується в промисловості. Головні риси мови значно розширюють сфери її застосування і забезпечують їй швидку імплементацію в архітектурах мереж.

Висновки до 2-го розділу

1. P4 - це мова програмування, призначена для опису процесу обробки пакетів в площині даних комутаторів, головними рисами якої є незалежність від цільової реалізації, незалежність від використовуваних протоколів та реконфігурованість полів.

2. Архітектура P4 описує набір програмованих блоків, компілятор та інтерфейси які використовуються для взаємодії з площиною управління. Програми написані для різних архітектур не сумісні між собою.
3. Додавання протоколів в цільовий пристрій P4 здійснюється за допомогою конструкції заголовків, в якій повністю описуються поля наявні в пакеті. Конвеєр обробки пакетів містить в собі таблиці та відповідні дії, які застосовуються до пакету за наявності збігу по таблицям. Всі таблиці та дії наявні в комутаторі описуються в програмі P4. Порядок застосування таблиць до пакету визначається програмістом і не є статичним.
4. Мова P4 все ще на ранній стадії прийняття, але уже використовується багатьма компаніями. Очікується що P4 стане стандартом для опису функціоналу площини пересилання даних в комутаторах SDN.

3. Мова програмування P4Runtime

3.1. Причини появи P4Runtime

Мова програмування P4 стала популярним способом визначення функціоналу площини управління в комутаторах. Але управління більшістю елементами площини пересилання даних таких як, додавання і видалення записів в таблицях відбувалося з використанням API який створений компілятором. Головним недоліком такого API є, прив'язка до програми. Зрештою, розробники мови P4, заявили що специфікація мови P4 не включає API рівня управління, надаючи програмістам та виробникам можливість створювати власні API. Це неминуче призвело до появи великої кількості несумісних API та додатків. Різні API не можуть бути перенесені в інші системи навіть якщо базові елементи пересилання визначені з відкритому і незалежному від виробника P4.

Табл. 2 Порівняння різних API

API	Target-independent	Protocol-independent
P4 compiler auto-generated	YES	NO
BMv2 CLI	NO	YES
OpenFlow	YES	NO
SAI	YES	NO

В таблиці 2 наведено порівняння різних API. API який згенерований компілятором P4, надає такі функції як додавання та видалення правил в таблиці. Але даний API прив’язаний до програми P4, тобто якщо програма міняється, міняється і API. Це приводить до неминучого перезавантаження контролеру.

BMv2 CLI не залежить від програми це означає, що API не змінюється, коли змінюється програма P4. Однак цей API прив'язаний до конкретного цільового пристрою і не може бути перенесений на інші цільові пристрої.

OpenFlow не залежить від цільового пристрою та архітектури. Тобто можна використовувати комутатори або мережеве обладнання від різних виробників за умови що воно підтримує протокол OpenFlow. Однак набір мережевих протоколів, які підтримує OpenFlow, визначений в специфікаціях та змінити його власноруч неможливо.

Switch Abstraction Interface (SAI) також як і протокол OpenFlow не має прив’язки до конкретного пристрою але все ще має фіксований набір мережевих протоколів які він підтримує.

Для вирішення питання сумісності API розробниками мови P4 було створено стандартизований, відкритий та апаратно незалежний (для пристроїв з фіксованою і програмованою функціональністю) API, який надає можливість контролювати роботу рівнів пересилання. Даний API отримав назву мова P4Runtime [10].

3.2. P4Runtime

Мова програмування P4Runtime – це API який створений розробниками мови P4, як стандартизований інтерфейс для взаємодії площини управління з площиною пересилання даних в комутаторах яких функціонал площини пересилання даних визначається програмою P4.

На рисунку 3.1 зображено інтерфейс P4Runtime. Він представлений в вигляді клієнт серверу. Серверна частина представлена в якості контролеру SDN, в свою чергу комутатор виступає в ролі клієнту зв'язок між якими відбувається за допомогою сервісу gRPC [12], що означає, що існує відповідний набір буферів протоколів (protobufs) [13], які визначають методи API та підтримувані параметри кожного з них. Разом вони реалізують зв’язок P4Runtime між контролером і комутатором.

При створенні даного зв’язку, спочатку відбувається описи функціоналу площини пересилання даних за допомогою P4. Після чого компілятор бере програму P4 та на основі наданої архітектури генерує на її основі двійкові файли, які завантажуються в кожний комутатор в мережі.

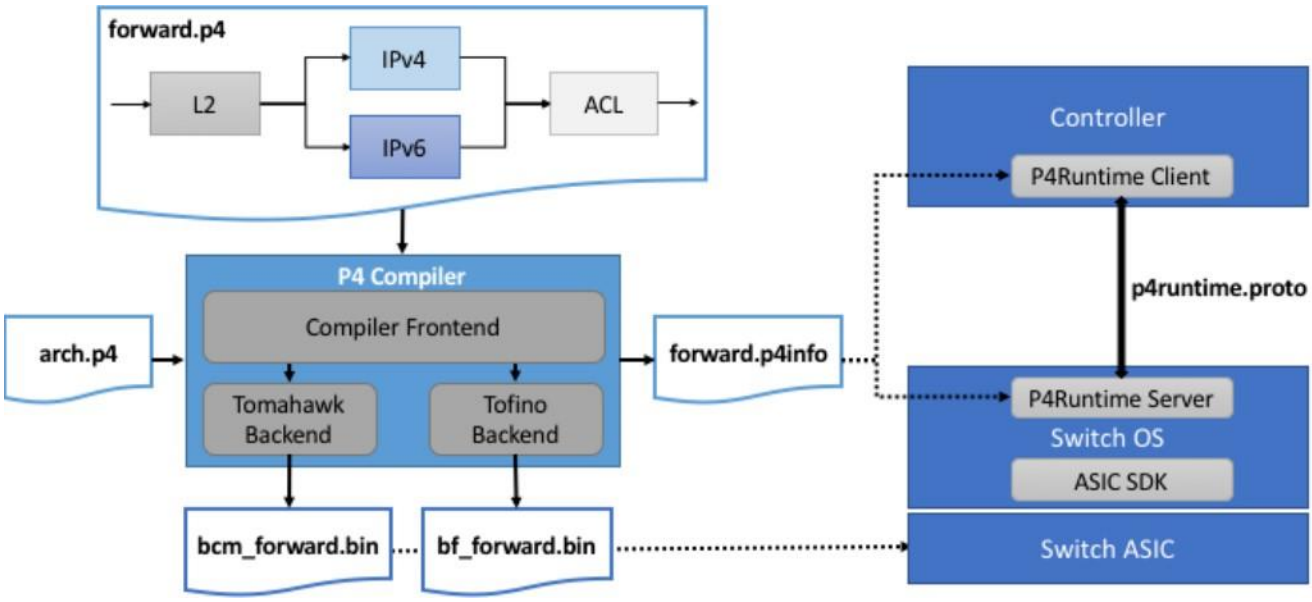

Рис. 3.1 Приклад генерації зв’язку між контролером та комутатором

Також компілятор генерує ще один двійковий файл в якому знаходиться опис всіх таблиць та відповідних дій які знаходяться в комутаторі. Даний файл використовується P4Runtime щоб надати контролеру інформацію про таблиці в комутаторі.

3.3. Гнучкість P4Runtime

P4Runtime може бути використаний для управління будь-яким комутатором, функціонал якого було визначено за допомогою мовою P4. Розробник може використовувати P4Runtime для управління існуючими комутаторами з фіксованим функціоналом площини пересилання даних, спершу написавши програму P4 для документування функціоналу комутатора за допомогою мови P4. Компілятор P4 автоматично визначає елементи, якими потрібно керувати, такі як таблиці, зазначені в програмі P4, для яких нам потрібно додавати та видаляти записи.

Як приклад, розглянемо програму P4 зліва внизу, яка включає таблицю найдовшого збігу префіксів (LPM) IPv4 рисунок 3.2. Коли комутатор запущений, нам потрібно буде додавати та видаляти записи з таблиці LPM, наприклад 8-бітний префікс, показаний посередині. В очікуванні компілятор генерує загальну схему з правого боку. У цьому прикладі площина управління використовує схему protobuf для кодування конкретного 8-бітового префікса, який вона хоче додати до таблиці.

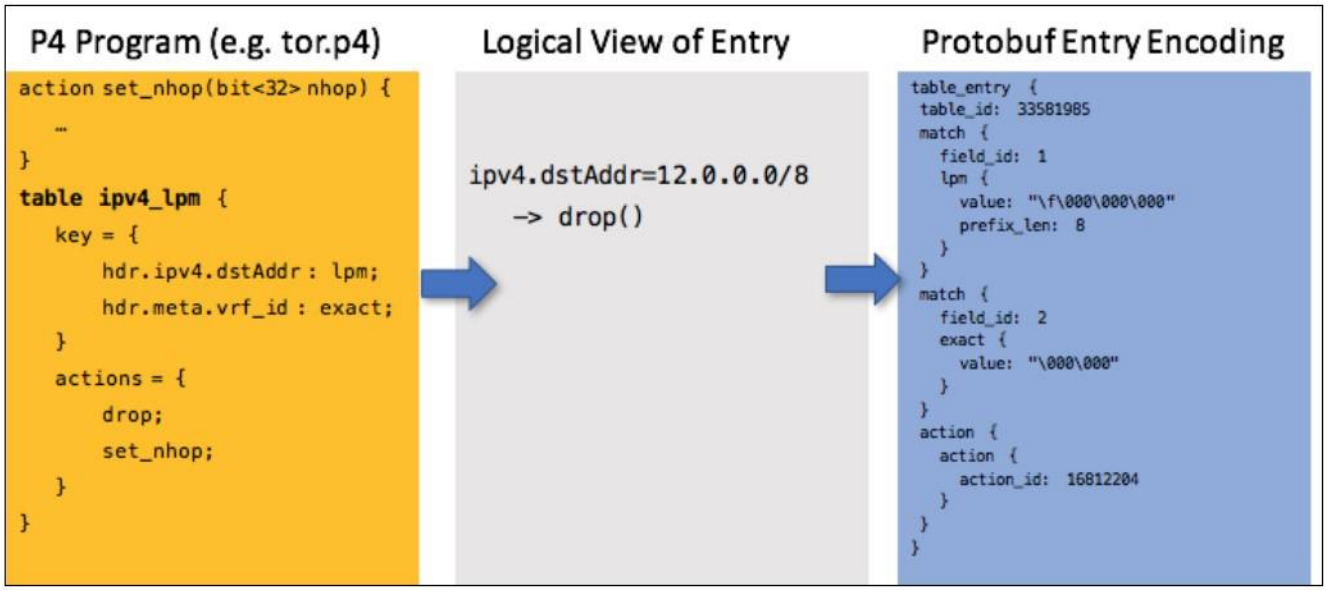

Рис. 3.2 Приклад роботи P4Runtime

Якщо комутатор програмований, розробник може розширити програму P4, додавши нові таблиці, які контролюються під час виконання; наприклад, таблиця префіксів IPv6, таблиця правил ACL, VXLAN або власна нова таблиця, приватна для цієї мережі. Для кожного випадку компілятор розширить схему - визначену як повідомлення protobuf - так, що новими таблицями можна керувати за допомогою P4Runtime.

3.4. Можливості P4Runtime

Оскільки P4Runtime був створений для безпосереднього використання в мовою програмування P4, вій надає багато можливостей яких відсутні в деяких API. Серед таких можливостей є:

- Арбітраж. Інтерфейс P4Runtime дозволяє підключити декілька контролерів до сервера P4Runtime на пристрої для забезпечення резервування та відмовостійкості. Наявність декількох контролерів дозволяє одному або декільком працювати в режимі резервного з перемиканням при відмові основного.

- Поділ ролей. P4Runtime підтримує спільне використання рівня даних P4 з різними головними (можливо і резервними) контролерами. Це робить можливими неоднорідні системи з комбінацією локального і одного або декількох віддалених рівнів управління (наприклад, локальний для L2 і віддалений для L3).
- Налаштовуваність рівня пересилання даних. Незалежна від конвеєра природа P4Runtime дозволяє «заштовхувати» абсолютно новий рівень пересилання (з іншим P4Info) через сам інтерфейс. Це дозволяє рівню управління використовувати програмованість рівня пересилання даних, коли вона підтримується.
- Введення та виведення пакетів. P4Runtime підтримує потокові пакети між клієнтами та серверами завдяки двосторонньому потоку gRPC. До пакету можна прив'язати виробничі метаданні (відповідно до P4Info).
- Звіти про помилки. P4Runtime забезпечує повнофункціональний механізм інформування клієнта про помилки. Кожне повідомлення містить канонічний код помилки і необов'язковий специфічний для пристрою код. Специфікація включає докладні рекомендації про використання канонічних кодів в різних ситуаціях, що може бути використано додатком для належного відгуку на помилки і розширення можливостей налагодження програм рівня управління.
- Можливість розширення архітектури. P4Runtime використовує будь-які повідомлень protobuf для підтримки довільних фірмових розширень, які можуть залишатися закритими. Нові повідомлення Protobuf можуть визначатися для настройки зовнішніх або потокових повідомлень між сервером і клієнтами.

3.5. Різниця між P4 та P4Runtime

Оскільки в назві обох міститься “P4”, деякі люди плутали P4Runtime з мовою P4. Насправді це цілком окремі проекти з відкритим кодом. Мова

програмування P4 використовується для визначення того, як комутатор обробляє пакети. По суті, P4 визначає конвеєр обробки пакетів в комутаторі. Мова P4 може бути використана для визначення функціоналу існуючого пристрою, або вона може бути використана для опису того, як програмований комутатор повинен обробляти пакети. Наприклад, мова P4 може бути використана для програмування чіпів комутації від Barefoot, розумних мережевих адаптерів від Netronome, FPGA від Xilinx і навіть для того, щоб сказати ядру, як обробляти пакети у Open vSwitch.

Мова P4Runtime - це API, що використовується для керування комутаторами, функціонал яких вже визначений мовою P4, незалежно від того, є цей комутатор з фіксованим функціоналом, напівпрограмованим чи повністю програмованим.

Висновки до 3-го розділу

В цьому розділі було розглянуто мову програмування P4Runtime, яка являє собою API створений розробниками мови P4, для взаємодії площини управління з площиною пересилання даних комутаторів SDN. P4Runtime це спроба замінити інтерфейси типу OpenFlow, та надати бажану гнучкість та сумісність для різних додатків.

4. Побудова мереж SDN з використанням мови P4.

4.1. Побудова мережі SDN на базі MININET

MININET

Mininet – це емулятор комп'ютерної мережі [14], який забезпечує середовище для проектування, тестування та відладки мереж SDN. Mininet дозволяє розробляти SDN на будь-якому ноутбуці чи ПК, а проекти SDN можуть безперешкодно переміщуватися між Mininet (що дозволяє недорогий і спрощений

розвиток) та реальним обладнанням, що працює за лінійною швидкістю в реальному розгортанні.

Mininet надає можливості:

- Швидке прототипування мереж SDN;
- Дозволяє декільком одночасним розробникам працювати самостійно над однією топологією;
- Дозволяє складне тестування топології без необхідності підключення фізичної мережі;
- Підтримує довільні користувацькі топології та включає базовий набір параметризованих топологій.

Mininet забезпечує простий спосіб отримати правильну поведінку системи (і, наскільки це підтримується вашим обладнанням, продуктивність) та експериментувати з топологіями. Мережі Mininet запускають реальний код, включаючи стандартні мережеві програми Unix / Linux, а також реальне ядро, та стек мережі Linux (включаючи будь-які розширення ядра). Через цей код, який ви розробляєте та тестуєте на Mininet, для контролера OpenFlow, модифікованого комутатора або хоста, може перейти до реальної системи з мінімальними змінами для реального тестування, оцінки продуктивності та розгортання. Це важливо, що дизайн, який працює в Mininet, зазвичай може переходити безпосередньо до апаратних комутаторів для переадресації пакетної швидкості.

Behavioral Model V2

Behavioral Model V2 (BMv2) – це основний інструмент для тестування та налагодження програм P4 [15]. BMv2 являє собою цільовий пристрій P4 (комутатор), здатний запускати програми P4, які націлені на архітектуру V1Model. Даний комутатор призначений лише для тестування програм P4, і використовувати його як програмний комутатор у промисловості не рекомендується.

Для компіляції програми P4 для BMv2 існує компілятор з відкритим кодом: p4c [16]. Інтерфейс між площиною управління та комутатором визначається за допомогою API Thrift RPC. Це дозволяє площині управління, наприклад, маніпулювати таблицями запущеної програми P4.

Apache Thrift

Apache Thrift - це бібліотека для побудови клієнтів та серверів віддаленого виклику процедур (RPC) на різних мовах програмування [17]. Як випливає з назви, RPC, Apache Thrift дозволяє клієнту віддалено викликати процедури на сервері, як якщо б вони були локальними. Метою проекту Thrift є забезпечення надійного та ефективного зв'язку між двома програмами, написаними різними мовами.

У Thrift усі типи даних та процедури визначаються в одному файлі з мовою визначення Thrift. Потім цей файл компілюється компілятором Thrift у код RPC на потрібній мові програмування.

4.2. Базове пересилання пакетів з використанням P4

Для того щоб облегшити написання програм P4, розробники мови створили вправи, які призначені для вивчення мови P4 [18]. В цих вправах надано готові топології для використання в Mininet з комутатором BMv2, так що головна задача програміста зводиться до правильного написання програми P4 під конкретну топологію. Тому метою цього розділу є написання програми P4, яка б реалізувала базове пересилання пакетів протоколу IPv4. Комутатори які будуть запрограмовані за допомогою P4 повинні виконувати наступні функції: оновлювати MAC-адреси джерела та пункту призначення; зменшувати TTL в заголовку пакету IPv4, та пересилати пакети на відповідний порт.

В даному випадку комутатори матимуть тільки одну таблицю, яка буде заповнена площиною управління статичними правилами. Кожне правило зіставляє IP-адресу, MAC-адресу та вихідний порт для наступного переходу. Правила площини управління визначені заздалегідь, тому потрібно реалізувати лише логіку пересилання в комутаторі з використанням P4.

На рисунку 4.1 зображено топологію для тестування базового пересилання пакетів. При запуску топології компілятор P4 скомпілює програму P4 та завантажить її в кожен із трьох комутаторів (s1, s2, s3) до кожного з яких підключено по одному хосту (h1, h2, h3), які мають відповідні IP адреси: 10.0.1.1, 10.0.2.2 та 10.0.3.3.

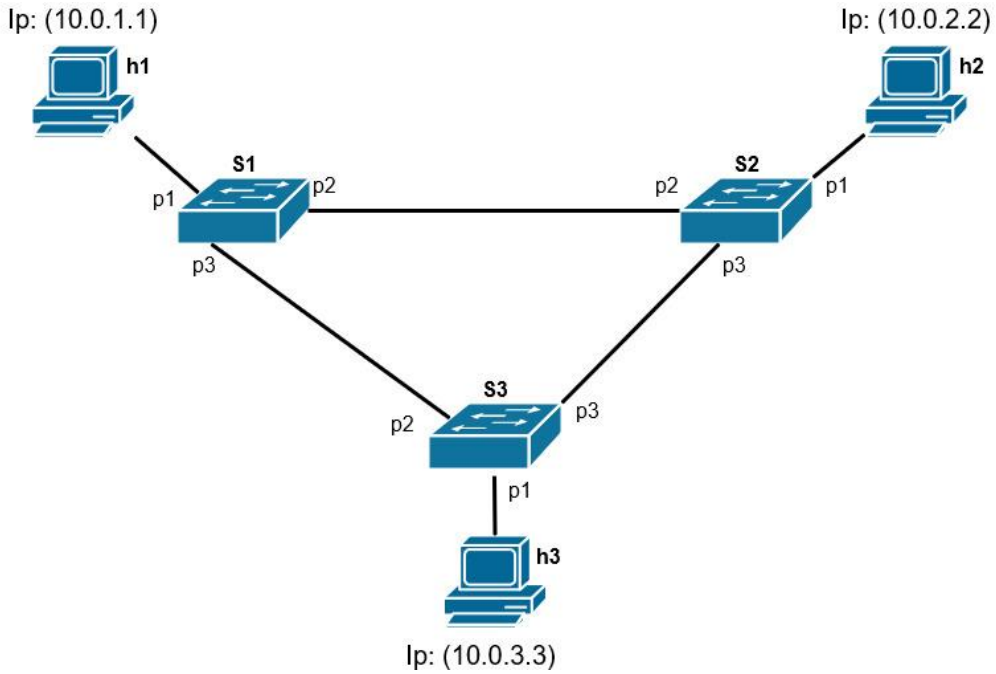

Рис. 4.1 Топологія для базового пересилання пакетів

Програма P4 для досягнення бажаного базового пересилання наведена нижче.

Програма P4 для базового пересилання пакетів IPv4.

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>
const bit<16> TYPE_IPV4 = 0x800;
```

```
typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}

header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}

struct metadata {
    /* empty */
}
struct headers {
    ethernet_t   ethernet;
    ipv4_t       ipv4;
}
parser MyParser(packet_in packet,
                out headers hdr,
```

```
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
```

```
    }
    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    apply {
        if (hdr.ipv4.isValid()) {
            ipv4_lpm.apply();
        }
    }
}
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}
control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
      update_checksum(
          hdr.ipv4.isValid(),
             { hdr.ipv4.version,
            hdr.ipv4.ihl,
               hdr.ipv4.diffserv,
               hdr.ipv4.totalLen,
               hdr.ipv4.identification,
               hdr.ipv4.flags,
```

```
                hdr.ipv4.fragOffset,
                hdr.ipv4.ttl,
                hdr.ipv4.protocol,
                hdr.ipv4.srcAddr,
                hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Після запуску топології видно що комутатори були сконфігуровані відповідно до поставленої задачі, появися командний рядок Mininet рисунок 4.2.

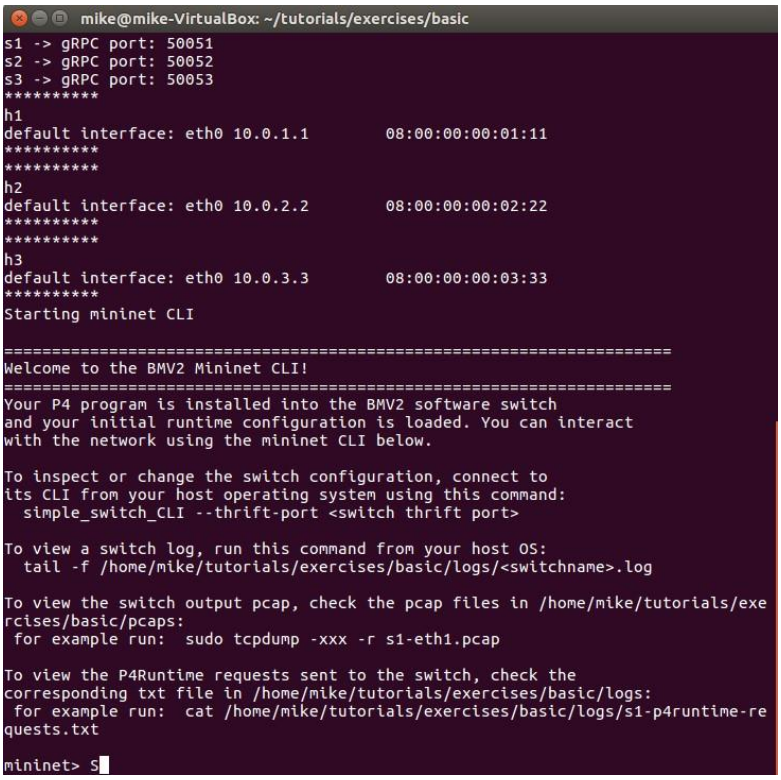

Рис. 4.2

Для перевірки досяжності хостів в топології скористаємося командами із Mininet, пінгування між двома хостами виконується за допомогою команди: mininet> h1 ping h2. На рисунку 4.3 показано що пакети доходять до пункту призначення, а отже програма P4 працює правильно.

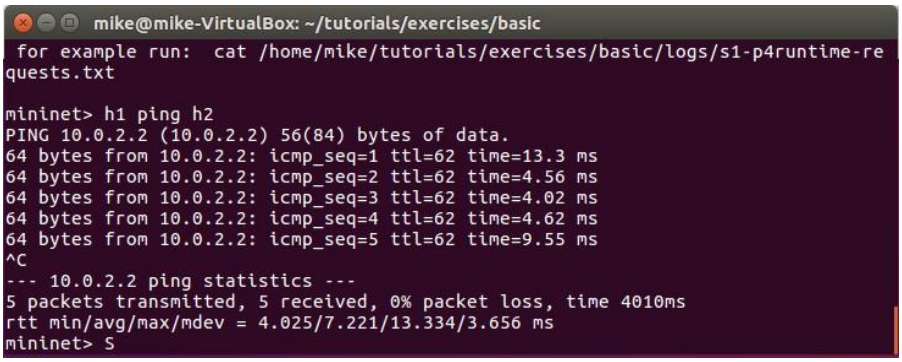

Рис. 4.3

4.3. Пересилання з використанням власного протоколу тунелю

В попередньому розділі було розглянуто базове пересилання пакетів на основі протоколу IPv4. В цьому ж розділі продемонстровано можливість визначати свої протоколи за допомогою заголовків P4.

Метою є додавання нового типу заголовка для інкапсуляції IP-пакету та модифікації коду комутатора, щоб замість цього він визначав порт призначення, використовуючи новий заголовок тунелю. Новий тип заголовка буде містити ідентифікатор протоколу, який вказує на тип пакета, що інкапсулюється, разом з ідентифікатором призначення, який буде використовуватися для маршрутизації.

Нижче наведена програма, яка реалізує пересилання з використанням тунелю. Дана програма написана на основі програми наведеної в розділі 4.2 , але в ній наявні дві таблиці оскільки було додано новий протокол з використанням заголовків P4. На рисунку 4.4 зображено топологію для використання протоколу тунелю.

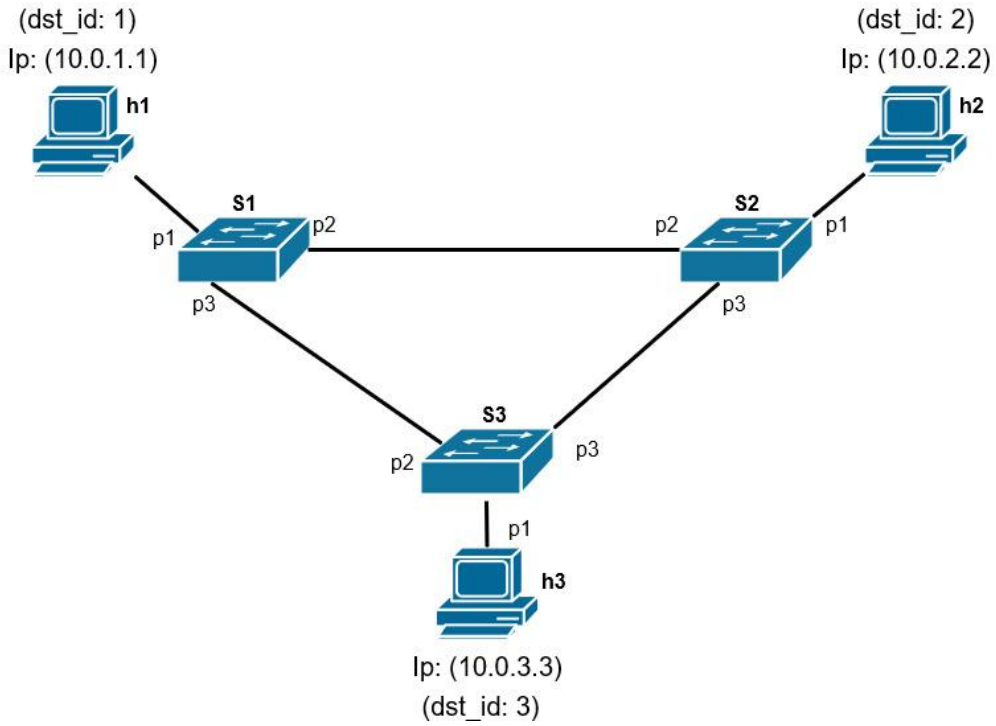

Рис. 4.4 Топологія для використання протоколу тунелю

Програма P4 для демонстрації роботи протоколу тунелю.

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

const bit<16> TYPE_MYTUNNEL = 0x1212;
const bit<16> TYPE_IPV4 = 0x800;

typedef bit<9>  egressSpec_t;
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16>   etherType;
}
header myTunnel_t {
    bit<16> proto_id;
    bit<16> dst_id;
}
header ipv4_t {
    bit<4>    version;
    bit<4>    ihl;
    bit<8>    diffserv;
    bit<16>   totalLen;
    bit<16>   identification;
    bit<3>    flags;
    bit<13>   fragOffset;
    bit<8>    ttl;
    bit<8>    protocol;
    bit<16>   hdrChecksum;
    ip4Addr_t srcAddr;
    ip4Addr_t dstAddr;
}
struct metadata {
```

```
    /* empty */
}
struct headers {
    ethernet_t   ethernet;
    myTunnel_t   myTunnel;
    ipv4_t       ipv4;
}
parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        transition parse_ethernet;
    }

    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_MYTUNNEL: parse_myTunnel;
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_myTunnel {
        packet.extract(hdr.myTunnel);
        transition select(hdr.myTunnel.proto_id) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
```

```
        transition accept;
    }
}

control MyVerifyChecksum(inout headers hdr, inout metadata meta) {
    apply {  }
}
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t standard_metadata) {
    action drop() {
        mark_to_drop(standard_metadata);
    }

    action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
        standard_metadata.egress_spec = port;
        hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = dstAddr;
        hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
    }
    table ipv4_lpm {
        key = {
            hdr.ipv4.dstAddr: lpm;
        }
        actions = {
            ipv4_forward;
            drop;
            NoAction;
        }
        size = 1024;
        default_action = drop();
    }

    action myTunnel_forward(egressSpec_t port) {
        standard_metadata.egress_spec = port;
```

```
    }

    table myTunnel_exact {
        key = {
            hdr.myTunnel.dst_id: exact;
        }
        actions = {
            myTunnel_forward;
            drop;
        }
        size = 1024;
        default_action = drop();
    }

    apply {
        if (hdr.ipv4.isValid() && !hdr.myTunnel.isValid()) {
            ipv4_lpm.apply();
        }

        if (hdr.myTunnel.isValid()) {
            myTunnel_exact.apply();
        }
    }
}
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t standard_metadata) {
    apply {  }
}
control MyComputeChecksum(inout headers  hdr, inout metadata meta) {
     apply {
      update_checksum(
          hdr.ipv4.isValid(),
             { hdr.ipv4.version,
            hdr.ipv4.ihl,
```

```
                hdr.ipv4.diffserv,
                hdr.ipv4.totalLen,
                hdr.ipv4.identification,
                hdr.ipv4.flags,
                hdr.ipv4.fragOffset,
                hdr.ipv4.ttl,
                hdr.ipv4.protocol,
                hdr.ipv4.srcAddr,
                hdr.ipv4.dstAddr },
            hdr.ipv4.hdrChecksum,
            HashAlgorithm.csum16);
    }
}
control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        packet.emit(hdr.ethernet);
        packet.emit(hdr.myTunnel);
        packet.emit(hdr.ipv4);
    }
}
V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Після запуску топології, було відкрито термінали на хостах 1 та 2 за допомогою команди: mininet> xterm h1 h2. В терміналі хосту 2 було ввімкнено відстеження пакетів за допомогою ./receive.py, рисунок 4.5.

Рис. 4.5

З хоста 1 було від правлено повідомлення за допомогою ./send.py на хост 2 з текстом student ITS. Дане повідомлення було прийняте на хості 2 рисунок 4.6.

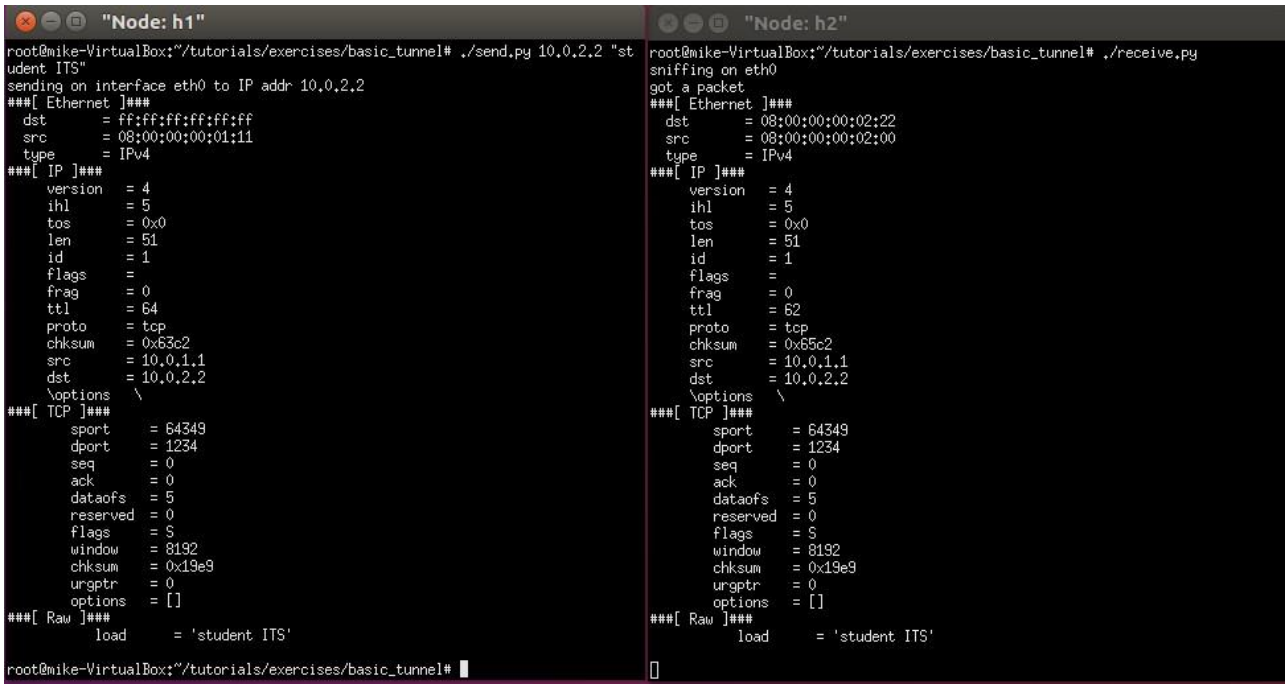

Рис. 4.6

Як можливо побачити з рисунка 4.6 для передачі було використано протокол IPv4, оскільки заголовок тунелю не було додано до пакету. Після чого було відправлено теж саме повідомлення, але уже з доданим заголовком тунелю.

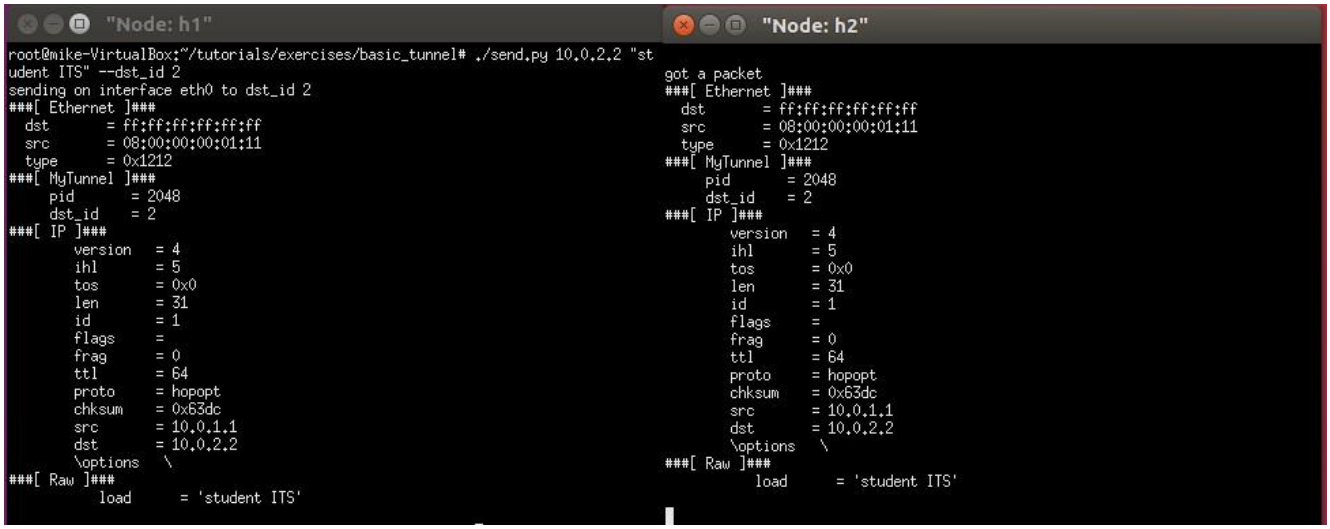

Рис. 4.7

Як видно з рисунка 4.7 при додаванні заголовку тунелю до пакету комутатори використали даний заголовок щоб перенаправити пакет в пункт призначення. Отже програма P4 працює правильно.

Висновки до 4-го розділу

Таким чином можна зробити висновок, що платформа Mininet та комутатор BMv2 надають можливість будувати програмно конфігуровані мережі з використанням мови P4. BMv2 надає можливість проводити відладку програми P4 для подальшого тестування в реальних мережах SDN.

Було продемонстровано визначення функціоналу комутатору за допомогою мови програмування P4. А також продемонстровано можливість додавання власних протоколів в комутатори.

Висновок

В дипломній роботі було розглянуто концепцію побудови програмно конфігурованих мереж, її основні елементи. Було досліджено архітектуру мережі SDN та її основні рівні. Більш детально було розглянуто взаємодію між рівнем управління та рівнем пересилання даних за допомогою протоколу OpenFlow. Розглянуто принцип роботи протоколу OpenFlow, принцип конвеєру обробки пакетів та недоліки OpenFlow в сучасних реаліях вимог до програмованості. Розглянуто новий підхід до програмування мікросхем.

Було досліджено причини появи мови програмування P4 та її головні риси. Розглянуто архітектуру мови P4 та основні елементи за допомогою яких визначається функціонал комутатору. Також було розглянуто мову P4Runtime яка виступає в якості інтерфейсу взаємодії між площиною пересилання даних та управління.

На сьогоднішній день чітко можна сказати, що програмно конфігуровані мережі стрімко розвиваються ставлячи нові вимоги до гнучкості та програмованості. P4 та P4Runtime були створені як відповідь на нові вимоги, вони все ще розвиваються але вже сьогодні надають користувачам можливості самим визначати функціонал площини пересилання даних комутаторів. В майбутньому очікується що P4 стане стандартом для програмування мережевих пристроїв мереж SDN.

Список використаних джерел

1. Berde P, Gerola M, Hart J, Higuchi Y, Kobayashi M, Koide T, Lantz B, O'Connor B, Radoslavov P, Snow W (2014) ONOS: towards an open, distributed SDN OS. In: Proceedings of the third workshop on Hot topics in software defined networking, ACM, pp 1–6
2. OpenFlow Specification version 1.5.1 [Электронный ресурс]. -URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf
3. C. Kozanitis, J. Huber, S. Singh, and G. Varghese, "Leaping multiple headers in a single bound: Wire-speed parsing using the Kangaroo system," in IEEE INFOCOM, pp. 830–838, 2010.
4. P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in ACM SIGCOMM, 2013.
5. Новий підхід до програмування [Электронный ресурс]. -URL: https://codilime.com/p4-network-programming-language-what-is-it-all-about/
6. Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. "P4: Programming protocol-independent packet processors". In:ACM SIGCOMM Computer Communication Review 44.3 (2014), pp. 87–95
7. P4 Specification version 1.2.2 [Электронный ресурс]. -URL: https://p4lang.github.io/p4-spec/docs/P4-16-v1.2.2.pdf
8. Формат пакету IP [Электронный ресурс]. -URL: https://zametkinapolyah.ru/kompyuternye-seti/4-3-struktura-i-zagolovok-ip-paketa-v-protokole-ipv4.html
9. SmartNIC [Электронный ресурс]. -URL: https://codilime.com/smartnics-with-p4-support/
10. P4Runtime [Электронный ресурс]. -URL: https://p4.org/p4/p4runtime-v1-release

11. P4Runtime Specification version 1.3.0 [Электронный ресурс]. -URL: https://p4lang.github.io/p4runtime/spec/v1.3.0/P4Runtime-Spec.pdf
12. gRPC [Электронный ресурс]. -URL: https://book.systemsapproach.org/e2e/rpc.html#grpc
13. ProtoBufs [Электронный ресурс]. -URL: https://book.systemsapproach.org/data/presentation.html#protobufs
14. Mininet [Электронный ресурс]. -URL: http://mininet.org/overview/
15. BMv2 [Электронный ресурс]. -URL: https://github.com/p4lang/behavioral-model
16. P4c компілятор [Электронный ресурс]. -URL: https://github.com/p4lang/p4c
17. Aditya Agarwal Mark Slee and Marc Kwiatkowski. Thrift: Scalable Cross Language Services Implementation. https://thrift.apache.org/static/files/thrift20070401.pdf
18. Tutorials P4 [Электронный ресурс]. -URL: https://github.com/p4lang/tutorials